AMATH 582 Homework 4

William Livernois (willll@uw.edu)

March 10, 2020

Abstract

Principle Component Analysis (PCA) and various classification techniques were applied to the MNIST database to differentiate and identify handwritten digits from images. This study used linear discriminant analysis (LDA), support vector machines (SVM), and decision trees to classify the images in a compressed principle components vectorspace.

1 Introduction and Overview

The MNIST (Modified National Institute of Science and Technology) database is a collection of images of handwritten digits captured from American highschool students and Census Bureau employees [4]. This database has been used to construct and test various machine learning algorithms over the past few decades, using the error rate as a benchmark for classifiers. Each handwritten digit is represented by a 28x28 greyscale image in the 60,000 image training set and 10,000 image test set. In this study, we have combined PCA with classifers to demonstrate the accuracy of machine learning techniques.

To start the analysis, PCA was applied to the training data to determine the major features present in the images. The most prominent features were then used as the dimensions for the classifiers. The LDA classifer was used on pairs of digits to determine the most distinguishable and least distinguishable digits from the data set. These digit pairs were used to compare the performance of the LDA, SVM and decision tree classifiers.

2 Theoretical Background

PCA was conducted on flattened images from the training set using singular value decomposition (SVD). With SVD the experimental matrix can be separated into the product $X = U\Sigma V^T$, with rotation matrices U and V^T, and diagonal singular value matrix Σ. This transformation allows us to determine the principal component matrix $Y = U^T X = \Sigma V^T$ that has a co-variance matrix given by

$$\begin{aligned} Cov[Y] &= \frac{1}{n-1} YY^T = \frac{1}{n-1}(U^T X)(U^T X)^T = \frac{1}{n-1}(U^T U\Sigma V^T)(U^T U\Sigma V^T)^T \\ &= \frac{1}{n-1}(\Sigma V^T)(\Sigma V^T)^T = \frac{1}{n-1}(\Sigma V^T V\Sigma^T) = \frac{\Sigma^2}{n-1} \end{aligned} \tag{1}$$

which must be diagonalized because Σ is a diagonal matrix[2]. In the case of these experiments, the principal components corresponded to the eigen-images of greatest variance (where the variance is proportional to Σ^2). By only using the top few principle components, the number of dimensions (28x28=784) can be drastically reduced to enable faster/better classification. The "rank" of the eigen-image space was determined visually, based on image reconstruction as shown in Figure 1. Based on these results, the top 10 principle components were chosen to represent the image, meaning only the first 10 columns of the U transformation matrix were used, which represents the transformation from a 784 pixel image to a 10-dimensional vector space. This similarly corresponds to the first 10 rows of the V transformation matrix, which represents the 10 eigen-image weights for each of the 60,000 training images.

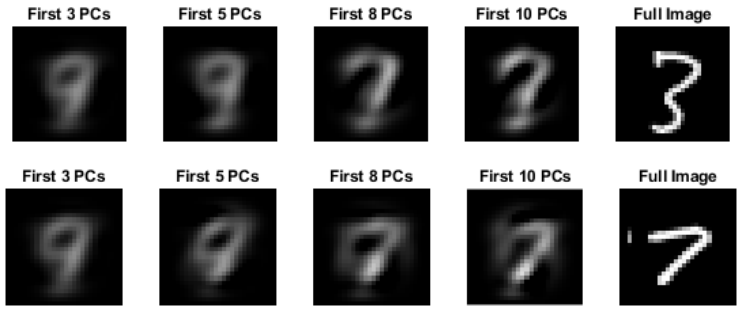

Figure 1: Comparisons of a handwritten 3 (top) and 7 (bottom) represented by 3, 5, 8, and 10 principle components with the original image on the right.

After transformation of each image into the principle component space, several supervised classification methods were used to predict the numbers shown by the images in the test data set. First, linear discriminant analysis was applied to pairs of digits to determine most distinguishable and least distinguishable pairs of handwritten digits. This algorithm was also applied to 3 randomly chosen digits for visualization and comparison to other techniques. Linear discriminant analysis is a finds a sub-plane (n-1 dimensional plane given n dimensions) that maximizes the variance between the projections of the data sets, as shown in Figure 2. This method can be solved as a linear transformation that maximizes the difference in the mean values of each class (given by μ_i) to determine the optimal projection.

The result from LDA was compared to a decision tree classifier and support vector machine (SVM) classifer. Decision tree classifiers optimize a series of binary thresholds to separate the data into groups, with the potential to create highly non-linear decision boundaries using many nodes. Support vector machines also require solving an optimization problem to maximize the distances between clusters of data based on a specified kernel. A decision boundary is generated based on the distance function (specified by the kernel) from the "support vector" points at the edge of each cluster, adding in a cost function during training for any wrongly classified points. All classifier training was done using the complete training set and tested/compared using a randomized subset of the testing data. This was used to cross-validate each classifier, ensuring reproducibility of results and ergodicity of the classfier.

3 Algorithm Implementation and Development

Principal components were extracted using `svd()` MATLAB function that uses a proprietary algorithm that is based on the `DGESVD` function in the LAPACK library[1]. The images from the dataset were centered and converted to grayscale, the only modification that was needed was flattening each image to a vector and converting to a double data type. In usual PCA analysis the data is centered around the mean, but in this

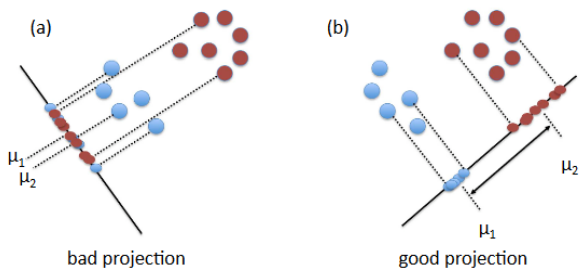

Figure 2: Visualization of Linear Discriminant Analysis (LDA) with a 2-dimensional data set. In this case the left projection would emerge and be used to generate the decision boundary for the data set

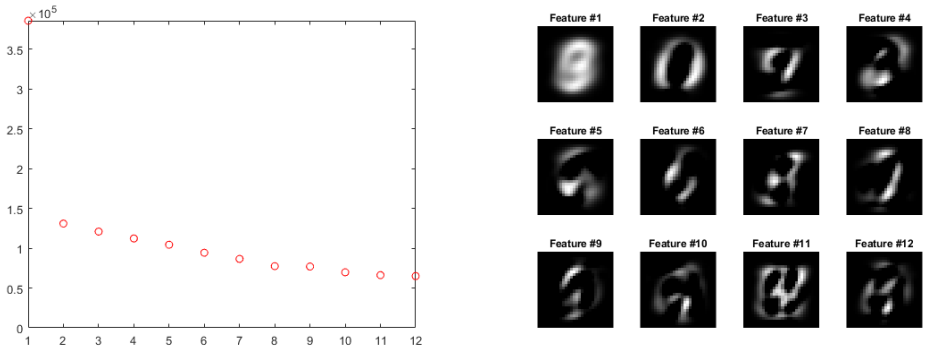

Figure 3: First 12 eigen-images from the singular value decomposition (right) as well as the corresponding stretching coefficients (left)

implementation the highest eigenvalue component represented this mean and was excluded from the principle components. The mean feature (Feature #1) and following first 11 principle components eigen-images are shown in Figure 3 as well as their Σ weights. The cropped U and $\Sigma \times V^T$ matrices were stored to a file for use as a transformation matrix and basis for all of the classifiers.

Cross-validation of the classifers was conducted using a random sampling from the dataset. First, a list of shuffled indices from 1 to 10,000 was generated representing all of the possible images in the test dataset. Then, the first 6000 indices were used to generate the test image matrix and test label vector. Each classifier was run with this test set and the rate of total incorrect matches was used to determine the error rate. An averaged error rate was used to compare the effectiveness of each classifier.

4 Computational Results

The first 10 principle components of the handwritten digits were extracted from the dataset, and a 3D visualization of the data projected onto three principle components (colored by label) is shown in Figure 4 left. This representation clearly shows the separation in clusters between the handwritten 2's and 4's just in this 3-component vector space. The LDA classifier was applied to all possible combinations of numbers and it was determined that the handwritten 4's and 9's were hardest to distinguish with an error rate of 19.0%. The most distinguishable digits using the LDA classifier were determined to be 0 and 1 which had an error rate of 0.3%. A three-way LDA classifier for digits 6, 8, and 9 had an error rate of 0.1%. A visualization of

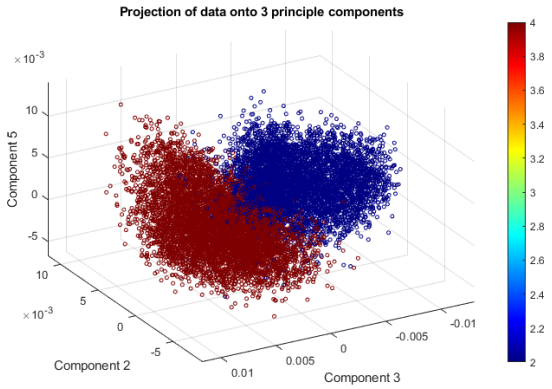

Figure 4: (left) A 3D projection of handwritten 2's and 4's on three principle components

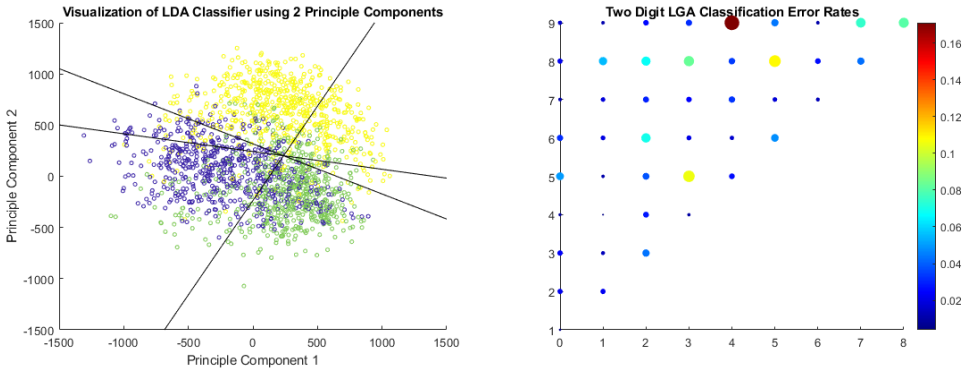

Figure 5: (left) A visualization of the three-way LDA classifer using 3 principle components and (right) the error rates for every combination of digits using a two-way LDA classifer

the LDA classifer using only 2 principle components can be seen in Figure 5 left and the error rates for all possible combinations of digits are plotted in Figure 5 right.

Three different classifiers (LDA, decision tree, and SVM) were trained to sort all of the 10 handwritten digits. From three runs using a random sampling of the test data the LDA classifer had an average error rate of 24.0% while the unrestricted decision tree classifier had an average error rate of 16.8% and the SVM classifier had an average error rate of 16.7%. The decision tree classifier was very large (over 6000 nodes) and had a slightly higher overall error rate but ran much faster than the SVM classifier. The SVM classifer took the longest time to train and had the longest runtime for cross-validation after training.

Finally, the SVM and decision tree classifiers were used to classify the most and least distinguishable digits (binary classification, average error rate over 3 trials reported). For the least distinguishable pair, 4 and 9, the SVM classifier was only slightly better than the LDA classifier giving an error rate of 18.7% while the decision tree classifier improved this to 14.9%. For the most distinguishable digits, 0 and 1, the decision tree classifier reduced the error rate to 0.13% and the SVM classifier further reduced this to 0.05%. It was clear that though the classifier generation took a significantly longer time the resulting classifier's had a significantly lower error rate than the simpler LDA classifier.

5 Summary and Conclusions

Using the training set to generate the principle components for classification significantly decreased the number of degrees of freedom (from 784 pixels to 10 dimension) and resulted in relatively accurate image classification. This is quite a powerful result because it shows that the images can be compressed significantly using the principle components and can be classified with reasonable accuracy. The decision tree and SVM classifiers generated showed overall improved accuracy over the basic LDA classifier, although the training time was much longer. The decision tree classifiers overall had the best results, with very similar error rates as the SVM classifier at a much smaller training and classification runtime.

The SVM classifier could have been improved significantly by modifying some of the model options. By default, it separates each of the classes into pairs for binary classification (one-versus-one or OvO type classifier) and used a hamming distance kernel to generate the decision surface. Other options include using a polynomial, exponential, or radial basis function (Gaussian) kernel. On the MNIST dataset website they have specified that Lecun et. al generated an SVM classifier using a Gaussian kernel with an error rate of 1.4%, which is significantly lower than the generated classifier from this project[3]. However, they used a more complete basis that focused on image features rather than just examining the principle components. Regardless, although there are even newer state-of-the-art techniques for image classification (such as convolutional neural networks) we have demonstrated the ability to classify images quickly with relatively high

precision using simple machine-learning models.

References

[1] Edward Anderson et al. *LAPACK users' guide*. SIAM, 1999.

[2] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013.

[3] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[4] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. *THE MNIST DATABASE of handwritten digits*. URL: http://yann.lecun.com/exdb/mnist/ (visited on 03/04/2021).

Appendix A MATLAB Functions

The MATLAB functions used for these calculations are listed below:

- `[u, s, v] = svd(X)` returns the rotation matrices `u` and `v` and the diagonal matrix `s` as the singular value decomposition of `X` such that `X=u*s*v'`
- `m = diag(M)` returns the diagonal vector `m` of matrix `M`
- `B = reshape(A, sz1, ... , szn)` reshapes the matrix `A` into an `sz1*...*szn` matrix
- `s = randperm(n)` returns a vector `s` of the first `n` randomly sorted non-zero integers
- `bool = ismember(v, group)` returns a boolean vector `bool` with the same dimensions `v` set to 1 if the element is a part of `group` and 0 if not.
- `A = nchoosek(v, n)` returns a matrix `A` with rows consisting of all possible combinations of `v` taking `n` elements at a time
- `pre = classify(xtrain, xtrainlabels, xtest, 'linear')` runs LDA classifier trained from the data `xtrain` with labels `xtrainlabels` and run on `xtest` to output predicted labels `pre`
- `[pre, , , , coeff] = classify(xtrain, xtrainlabels, xtest, 'linear')` runs LDA classifier trained from the data `xtrain` with labels `xtrainlabels` and run on `xtest` to output predicted labels `pre` and a struct with the decision boundary coefficients given by `coeff`
- `tree = fitctree(xtrain, xtrainlabels, 'CrossVal', 'On')` generates a decision tree classifier object `tree` trained using data `xtrain` with labels `xtrainlabels` that is cross-validated on the training data
- `svm= fitcecoc(xtrain, xtrainlabels)` generates an SVM classifier object `svm` trained using data `xtrain` with labels `xtrainlabels` using the default hamming distance kernal and combining one-vs-one binary classifiers for more than two groups (labels)
- `pre= fitcecoc(class, xtest)` runs classification of `xtest` using the classifier object `class` outputing label predictions into list `pre`
- `tic; [COMMANDs]; toc;` prints the runtime to execute the given commands.

Functions used for basic arithmetic or plot formatting are not included above, but the important figures are included in the Computation Results section above.

Appendix B MATLAB Code

All code is located in the "Homework 4" folder of the github repository (`https://github.com/wliverno/AMATH582`).

Several MATLAB files were used for this project, the first one being `mnistSVD.m` which generates the principle components from the training set, runs some visualizations, and stores the matrices to a file:

```
clear all, close all;

%Collect images/labels and randomly sample first nsample images
[images, labels] = mnist_parse('train-images.idx3-ubyte', 'train-labels.idx1-ubyte');
[lx, ly, n] = size(images);
nsample = 60000;
sample = randperm(n);
sample = sample(1:nsample);

% Reshape images to make data matrix and run SVD
A = zeros(lx*ly, nsample);
for i=1:nsample
    ind = sample(i);
    A(:,i) = reshape(double(images(:,:,ind)), lx*ly, 1);
end
[U, S, V] = svd(A, 'econ');
s=diag(S);

%Plot Singular Values and eigen-images for the first 12 features
plot(s(1:12), 'ro');
axis([1, 12, 0, max(s)+1])
figure;
for i=1:12
    pc = U(:,i)*255/max(U(:,i));
    im = uint8(reshape(pc, lx, ly));
    subplot(3,4,i), imshow(im), title(['Feature #', num2str(i)]);
end

%Extract [features] principle components
features = 2:11;
svPC = S*V';
svPC = svPC(features, :);
uPC = U(:,features);
imgPC = uPC*svPC;

%Show rank given different principle components
figure;
rank=[3, 5, 8, 10, 784];
for i=1:length(rank)
    im = uint8(reshape(U(:,1:rank(i))*S(1:rank(i),:)*(V(1,:)'), lx, ly));
    subplot(1,5,i), imshow(im), title(['First ', num2str(rank(i)), ' PCs']);
end
title('Full Image');

%3D plot showing projection on principle components of each number
components = [2 3 5];
C = labels(sample);
```

```
sublist = 1:nsample;
sublist = ((C==2|C==4)); %To select certain number values for visualization
X = V(sublist, components(1));
Y = V(sublist, components(2));
Z = V(sublist, components(3));
Sz = ones(length(X), 1)*10;
C = C(sublist);
figure;
scatter3(X,Y,Z,Sz,C), colormap('jet'), colorbar;
title('Projection of data onto 3 principle components');
xlabel(['Component ', num2str(components(1))]);
ylabel(['Component ', num2str(components(2))]);
zlabel(['Component ', num2str(components(3))]);

save('pcaMats', 'U','S','V','uPC', 'svPC')
```

The file `mnistClassifiers.m` is a function that reads the principle component matrices and uses them to run the classifer training, printing reports and creating visualizations:

```
clear all, close all;

load pcaMats.mat;

%Collect training sets and take random sample
[trainimages, trainlabels] = mnist_parse('train-images.idx3-ubyte', 'train-labels.idx1-ubyte');
[lx, ly, n] = size(trainimages);
trainPC = getPCABasis(trainimages, uPC);

%Collect test set
[testimages, testlabels] = mnist_parse('t10k-images.idx3-ubyte', 't10k-labels.idx1-ubyte');
[lx, ly, n] = size(testimages);

%Use randomized subset for cross-validation between runs
nsample = 6000;
sample = randperm(n);
sample = sample(1:nsample);
testlabels = testlabels(sample);
testimages = testimages(:, :, sample);
testPC = getPCABasis(testimages, uPC);

% imshow(uint8(reshape(uPC*testPC(:,22), lx,ly)))
% testlabels(22)
% figure;

%Set up set for LDA on two digits, specified by split1, split2
combos = nchoosek(0:9, 2);
probs = zeros(length(combos(:,1)), 1);
for i=1:length(combos)
    subset=ismember(trainlabels, combos(i,:));
    trainPC2 = trainPC(:, subset);
    trainlabels2 = trainlabels(subset);

    subset=ismember(testlabels, combos(i,:));
    testPC2 = testPC(:, subset);
```

```
    testlabels2 = testlabels(subset);

    %Linear Classification
    pre=classify(testPC2', trainPC2', trainlabels2, 'linear');
    probs(i) = sum(pre==testlabels2)/length(pre);
end
% Calculate best and worst classifications
[minP, indMin] = min(probs);
fprintf('The worst classification with LDA was between %i and %i with %.1f%% incorrect matches \n', comb
[maxP, indMax] = max(probs);
fprintf('The best classification with LDA was between %i and %i with %.1f%% incorrect matches \n', combo
% Plot LDA probabilities
figure;
scatter(combos(:,1), combos(:,2), (1-probs)*900, (1-probs), 'filled'), colormap('jet'), colorbar;
title('Two Digit LGA Classification Error Rates');

%Set up LDA for 3 digits
digits = [6 8 9];
subset=ismember(trainlabels, digits);
trainPC3 = trainPC(:, subset);
trainlabels3 = trainlabels(subset);

subset=ismember(testlabels, digits);
testPC3 = testPC(:, subset);
testlabels3 = testlabels(subset);

%Linear Classification and visualization using first two PC's
[pre, ~, ~, ~, coeff]=classify(testPC3(1:2,:)', trainPC3(1:2,:)', trainlabels3, 'linear');
figure, scatter(testPC3(1, :), testPC3(2, :), ones(1,length(testlabels3))*10, testlabels3');
lim = [-1500 1500 -1500 1500];
hold on;
matcoord = [1, 2; 1, 3; 2, 3];
for i = 1:3
    K = coeff(matcoord(i,1),matcoord(i,2)).const;
    L = coeff(matcoord(i,1),matcoord(i,2)).linear;
    f = @(x,y) K + L(1)*x + L(2)*y;
    h = fimplicit(f,lim);
    set(h,'Color','k')
end
title('Visualization of LDA Classifier using 2 Principle Components');
xlabel('Principle Component 1')
ylabel('Principle Component 2')

%Three Digit LDA with all PCs
pre=classify(testPC3', trainPC3', trainlabels3, 'linear');
threeDigitErr = 1-(sum(pre==testlabels3)/length(pre));
fprintf('Three digit LDA classification between %i, %i, and %i had an error rate of %.1f%% \n', digits(

%LDA 10-digit classifier
disp('Running 10-digit LDA Classifier...');
tic;
pre=classify(testPC', trainPC', trainlabels, 'linear');
toc;
ldaErr = 1-(sum(pre==testlabels)/length(pre))
```

```
%Decision Tree 10-digit Classifier
tree=fitctree(trainPC',trainlabels,'CrossVal','on');%'MaxNumSplits',200,
%view(tree.Trained{1},'Mode','graph');
%dtErr = kfoldLoss(tree)
disp('Running 10-digit Decision Tree Classifier...');
tic;
pre = predict(tree.Trained{1}, testPC');
toc;
treeErr = 1-(sum(pre==testlabels)/length(pre))

%SVM 10-digit Classifier
trainPCSVM = trainPC/max(trainPC(:));
testPCSVM = testPC/max(testPC(:));
svm = fitcecoc(trainPCSVM', trainlabels);
disp('Running 10-digit Decision Tree Classifier...');
tic;
pre = predict(svm, testPCSVM');
toc;
svmErr = 1-(sum(pre==testlabels)/length(pre))

%SVM and Decision Tree for most distinguisable digits
subset=ismember(trainlabels, combos(indMax,:));
trainPC2 = trainPC(:, subset);
trainPC2SVM = trainPC2/max(trainPC2(:));
trainlabels2 = trainlabels(subset);

subset=ismember(testlabels, combos(indMax,:));
testPC2 = testPC(:, subset);
testPC2SVM = testPC2/max(testPC2(:));
testlabels2 = testlabels(subset);

disp('Generating Decision Tree and SVM for most distinguishable digits:')
tree=fitctree(trainPC2',trainlabels2,'CrossVal','on');
pre = predict(tree.Trained{1}, testPC2');
treeErrBestDigits = 1-(sum(pre==testlabels2)/length(pre))
svm = fitcecoc(trainPC2SVM', trainlabels2);
pre = predict(svm, testPC2SVM');
svmErrBestDigits = 1-(sum(pre==testlabels2)/length(pre))

%SVM and Decision Tree for least distinguisable digits
subset=ismember(trainlabels, combos(indMin,:));
trainPC2 = trainPC(:, subset);
trainPC2SVM = trainPC2/max(trainPC2(:));
trainlabels2 = trainlabels(subset);

subset=ismember(testlabels, combos(indMin,:));
testPC2 = testPC(:, subset);
testPC2SVM = testPC2/max(testPC2(:));
testlabels2 = testlabels(subset);

disp('Generating Decision Tree and SVM for least distinguishable digits:')
tree=fitctree(trainPC2',trainlabels2,'CrossVal','on');
pre = predict(tree.Trained{1}, testPC2');
```

```
treeErrWorstDigits = 1-(sum(pre==testlabels2)/length(pre))
svm = fitcecoc(trainPC2SVM', trainlabels2);
pre = predict(svm, testPC2SVM');
svmErrWorstDigits = 1-(sum(pre==testlabels2)/length(pre))


%Linear Classification
pre=classify(testPC2', trainPC2', trainlabels2, 'linear');
probs(i) = sum(pre==testlabels2)/length(pre);

function out = getPCABasis(in, transform)
    [lx, ly, n] = size(in);
    out = zeros(lx*ly, n);
    for i=1:n
        out(:,i) = reshape(double(in(:,:,i)), lx*ly, 1);
    end
    out = transform'*out;
end
```

The file `mnist_parse.m` is a helper function used to import the MNIST images from their file format:

```
function [images, labels] = mnist_parse(path_to_digits, path_to_labels)

% The function is curtesy of stackoverflow user rayryeng from Sept. 20,
% 2016. Link: https://stackoverflow.com/questions/39580926/how-do-i-load-in-the-mnist-digits-and-label-

% Open files
fid1 = fopen(path_to_digits, 'r');

% The labels file
fid2 = fopen(path_to_labels, 'r');

% Read in magic numbers for both files
A = fread(fid1, 1, 'uint32');
magicNumber1 = swapbytes(uint32(A)); % Should be 2051
fprintf('Magic Number - Images: %d\n', magicNumber1);

A = fread(fid2, 1, 'uint32');
magicNumber2 = swapbytes(uint32(A)); % Should be 2049
fprintf('Magic Number - Labels: %d\n', magicNumber2);

% Read in total number of images
% Ensure that this number matches with the labels file
A = fread(fid1, 1, 'uint32');
totalImages = swapbytes(uint32(A));
A = fread(fid2, 1, 'uint32');
if totalImages ~= swapbytes(uint32(A))
    error('Total number of images read from images and labels files are not the same');
end
fprintf('Total number of images: %d\n', totalImages);

% Read in number of rows
A = fread(fid1, 1, 'uint32');
```

```
numRows = swapbytes(uint32(A));

% Read in number of columns
A = fread(fid1, 1, 'uint32');
numCols = swapbytes(uint32(A));

fprintf('Dimensions of each digit: %d x %d\n', numRows, numCols);

% For each image, store into an individual slice
images = zeros(numRows, numCols, totalImages, 'uint8');
for k = 1 : totalImages
    % Read in numRows*numCols pixels at a time
    A = fread(fid1, numRows*numCols, 'uint8');

    % Reshape so that it becomes a matrix
    % We are actually reading this in column major format
    % so we need to transpose this at the end
    images(:,:,k) = reshape(uint8(A), numCols, numRows).';
end

% Read in the labels
labels = fread(fid2, totalImages, 'uint8');

% Close the files
fclose(fid1);
fclose(fid2);

end
```
